OpenCV

Chentai Kao

2/7/2014

Outline

- Why OpenCV?
- Installation
- C++ API
- Python API
- Demo (using Python API)

Why OpenCV?

Why OpenCV?

	Matlab	OpenCV
Ease of use	✓	
Speed		✓
Resource-saving		✓
Cost		✓
Memory management	✓	
Portability		✓
Debugging	✓	

OpenCV Highlights

- Focus on real-time image processing
- Written in C/C++
- C/C++ interface
  - Also in Python, Java, Matlab/Octave
- Cross-platform
  - Windows, Mac, Linux, Android, iOS, etc
- Use CMake
- Open source and free!

Applications

- Feature extraction
- Recognition (facial, gesture, etc)
- Segmentation
- Robotics
- Structure from motion
- Machine learning support
  - Boosting, k-nearest neighbor, SVM, etc

Modules and Functionality

- **core** Basic data structures
- **imgproc** Image processing, filter, transformation
- **highgui** GUI, codecs, image/video capturing
- **calib3d** Camera calibration, 3D reconstruction
- **feature2d** 2D feature (detector, descriptor, matching)
- **video** Motion tracking, foreground extraction
- **objdetect** Object detection (face, people)
- **ml** Machine learning library
- **gpu** GPU acceleration

Installation

Install on Windows

- Using pre-built libraries
  - Quick but less flexible
- Build from source code (recommended)
  - Download source code
  - Install an IDE (Visual Studio, codeblocks, etc)
  - Install CMake
  - Use CMake to configure and generate Makefile
  - Use IDE to build both DEBUG and RELEASE
- Add system path for DLL

Install on Linux

- Install GCC, CMake, ffmpeg, pkgconfig
- Download source code
- Use CMake to configure and generate Makefile
- Build OpenCV
  - make
  - make install

Install on Mac

- Install "homebrew"
- Use homebrew to install OpenCV
  - $ sudo brew install opencv
- Or use "macports" to install OpenCV
  - $ sudo port install opencv

C++ API

Basic Structures

- Point, Point2f, Point3f
  - Points specified by its coordinates
- Size
  - Specify the size of an image
- Vec, Vec3f
  - Describe multi-channel pixel values
- Mat
  - N-dimensional array, mostly used to store images

Point

- 2D or 3D point

```
Point2f a(0.3f, 0.f), b(0.f, 0.4f);
Point pt = (a + b)*10.f;
cout << pt.x << ", " << pt.y << endl;
```

- Operators: +, -, *, ==, !=
- Functions:
  - Point.dot(<Point>)
  - Point.inside(<Rect>)

Size

- Store matrix size (cols, rows)
  - Mat.size()
  - Mat.Mat(<Size>, <type>)

Vec

- Commonly used to describe pixel values

```
// allocate a 320x240 color image filled with green
Mat_<Vec3b> img(240, 320, Vec3b(0,255,0));
```

```
v1 = v2 + v3
v1 = v2 - v3
v1 = v2 * scale
v1 = scale * v2
v1 = -v2
v1 += v2 and other augmenting operations
v1 == v2, v1 != v2
norm(v1) (euclidean norm)
```

Mat

- Primary data structure in OpenCV
- rows, cols
- Primitive type
  - CV_<bit-depth>{U|S|F}C(<number_of_channels>)
  - CV_8UC1: uchar, 1 channel
  - CV_32FC3: floating-point, 3 channels (BGR)

```
// create a new 320x240 image
Mat img(Size(320,240),CV_8UC3);
```

How to Access Pixel Value

- Safer but slower way:

```
Mat H(100, 100, CV_64F);
for(int i = 0; i < H.rows; i++)
  for(int j = 0; j < H.cols; j++)
    H.at<double>(i,j) = 1./(i+j+1);
```

- Efficient way (but lose some readability):

```
for(int i = 0; i < H.rows; i++) {
  double* p = H.ptr<double>(i);
  for (int j = 0; j < H.cols; j++)
    p[j] = 1./(i+j+1);
```

How to Access Pixel Value (cont'd)

- Mat.data is another choice.
  - Warning: type of Mat.data is uchar*
  - Use MACRO to enhance readability.

```
#define FAT_3D(m, i, j, k) \
  (*((float*)((m).data) + \
  (i) * (m).cols * (m).channels() + \
  (j) * (m).channels() + \
  (k)))
```

  - Usage:

```
// Set the (10, 5, 1) element of img to 2.5
FAT_3D(img, 10, 5, 1) = 2.5;
```

Python API

Python Interface

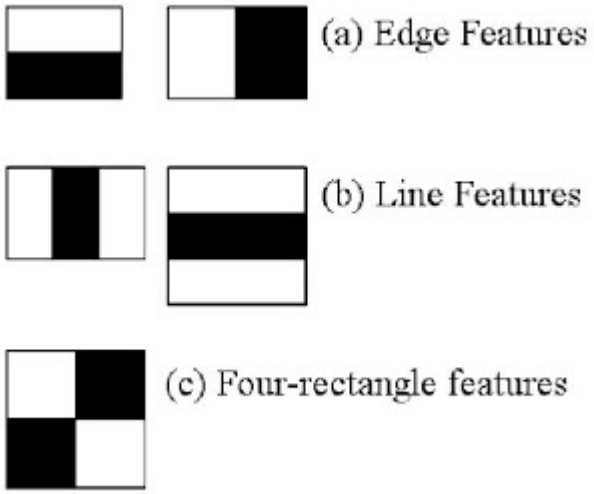
- Mostly the same as C++ interface
- Which interface, "cv" or "cv2"?
  - cv: same data type as in C++ (e.g. cvMat)
  - cv2: returns NumPy object (e.g. ndarray)
- In general, use "cv2"
  - More elegant, save several function calls

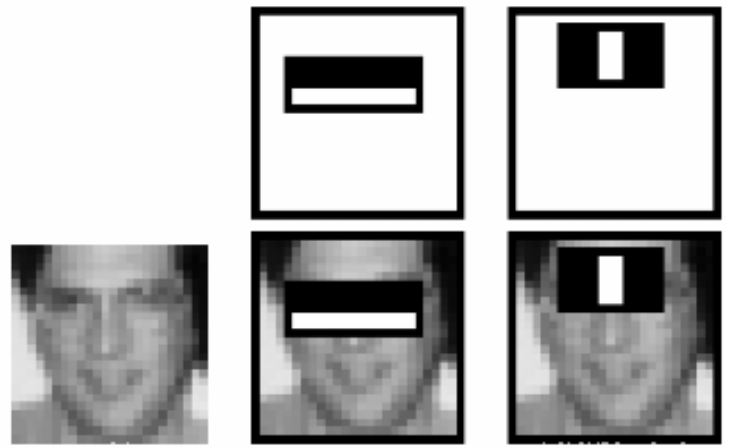
```
// pixel value at (i, j) position
m[i, j] = 1
// region of interest
ROI = m[c1:c2, r1:r2]
```

DEMO (using Python API)

Face Detection using Haar Cascades

- Webcam input, real-time display
- Haar cascades





```python
import numpy as np
import cv2

HAAR_CASCADE_PATH = "haarcascade_frontalface_default.xml"
CAMERA_INDEX = 0

if __name__ == "__main__":
    capture = cv2.VideoCapture(CAMERA_INDEX)
    faces = []

    count = 0
    key = -1

    face_cascade = cv2.CascadeClassifier(HAAR_CASCADE_PATH)
    while (key == -1):
        flag, image = capture.read()
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        # Only run the Detection algorithm every 5 frames to improve performance
        if count % 5 == 0:
            faces = face_cascade.detectMultiScale(gray, 1.3, 5)

        if faces is not None:
            for (x,y,w,h) in faces:
                cv2.rectangle(image,(x,y),(x+w,y+h),(255,0,0),2)

        cv2.imshow('image (press ESC to quit)',image)
        key = cv2.waitKey(10)
        count += 1

    cv2.destroyAllWindows()
```

Import modules

```
import numpy as np
import cv2

HAAR_CASCADE_PATH = "haarcascade_frontalface_default.xml"
CAMERA_INDEX = 0

if __name__ == "__main__":
    capture = cv2.VideoCapture(CAMERA_INDEX)
    faces = []

    count = 0
    key = -1

    face_cascade = cv2.CascadeClassifier(HAAR_CASCADE_PATH)
    while (key == -1):
        flag, image = capture.read()
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        # Only run the Detection algorithm every 5 frames to improve performance
        if count % 5 == 0:
            faces = face_cascade.detectMultiScale(gray, 1.3, 5)

        if faces is not None:
            for (x,y,w,h) in faces:
                cv2.rectangle(image,(x,y),(x+w,y+h),(255,0,0),2)

        cv2.imshow('image (press ESC to quit)',image)
        key = cv2.waitKey(10)
        count += 1

    cv2.destroyAllWindows()
```

Webcam input

```python
import numpy as np
import cv2

HAAR_CASCADE_PATH = "haarcascade_frontalface_default.xml"
CAMERA_INDEX = 0

if __name__ == "__main__":
    capture = cv2.VideoCapture(CAMERA_INDEX)
    faces = []

    count = 0
    key = -1

    face_cascade = cv2.CascadeClassifier(HAAR_CASCADE_PATH)
    while (key == -1):
        flag, image = capture.read()
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        # Only run the Detection algorithm every 5 frames to improve performance
        if count % 5 == 0:
            faces = face_cascade.detectMultiScale(gray, 1.3, 5)

        if faces is not None:
            for (x,y,w,h) in faces:
                cv2.rectangle(image,(x,y),(x+w,y+h),(255,0,0),2)

        cv2.imshow('image (press ESC to quit)',image)
        key = cv2.waitKey(10)
        count += 1

    cv2.destroyAllWindows()
```

Face detection

```
import numpy as np
import cv2

HAAR_CASCADE_PATH = "haarcascade_frontalface_default.xml"
CAMERA_INDEX = 0

if __name__ == "__main__":
    capture = cv2.VideoCapture(CAMERA_INDEX)
    faces = []

    count = 0
    key = -1

    face_cascade = cv2.CascadeClassifier(HAAR_CASCADE_PATH)
    while (key == -1):
        flag, image = capture.read()
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        # Only run the Detection algorithm every 5 frames to improve performance
        if count % 5 == 0:
            faces = face_cascade.detectMultiScale(gray, 1.3, 5)

        if faces is not None:
            for (x,y,w,h) in faces:
                cv2.rectangle(image,(x,y),(x+w,y+h),(255,0,0),2)

        cv2.imshow('image (press ESC to quit)',image)
        key = cv2.waitKey(10)
        count += 1

    cv2.destroyAllWindows()
```

Real-time display

```
import numpy as np
import cv2

HAAR_CASCADE_PATH = "haarcascade_frontalface_default.xml"
CAMERA_INDEX = 0

if __name__ == "__main__":
    capture = cv2.VideoCapture(CAMERA_INDEX)
    faces = []

    count = 0
    key = -1

    face_cascade = cv2.CascadeClassifier(HAAR_CASCADE_PATH)
    while (key == -1):
        flag, image = capture.read()
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        # Only run the Detection algorithm every 5 frames to improve performance
        if count % 5 == 0:
            faces = face_cascade.detectMultiScale(gray, 1.3, 5)

        if faces is not None:
            for (x,y,w,h) in faces:
                cv2.rectangle(image,(x,y),(x+w,y+h),(255,0,0),2)

        cv2.imshow('image (press ESC to quit)',image)
        key = cv2.waitKey(10)
        count += 1

    cv2.destroyAllWindows()
```

Feature Matching

- Features detection by ORB descriptors
- Use brute-force matching
- Show the best 10 matches

Read images

```
import numpy as np
import cv2

img1 = cv2.imread('church.jpg',0) # queryImage
img2 = cv2.imread('church_part.jpg',0) # trainImage

# Initiate SIFT detector
orb = cv2.ORB()

# find the keypoints and descriptors with SIFT
kp1, des1 = orb.detectAndCompute(img1,None)
kp2, des2 = orb.detectAndCompute(img2,None)

# create BFMatcher object
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)

# Match descriptors.
matches = bf.match(des1,des2)

# Sort them in the order of their distance.
matches = sorted(matches, key = lambda x:x.distance)

# Draw first 10 matches.
h1, w1 = img1.shape[:2]
h2, w2 = img2.shape[:2]
img3 = np.zeros((max(h1, h2), w1 + w2, 3), np.uint8)
img3[:h1, :w1, 0] = img1
img3[:h2, w1:, 0] = img2
img3[:, :, 1] = img3[:, :, 0]
img3[:, :, 2] = img3[:, :, 0]

for m in matches[:10]:
    # draw the keypoints
    # print m.queryIdx, m.trainIdx, m.distance
    color = tuple([np.random.randint(0, 255) for _ in xrange(3)])
    cv2.line(img3, (int(kp1[m.queryIdx].pt[0]), int(kp1[m.queryIdx].pt[1])) , (int(
kp2[m.trainIdx].pt[0] + w1), int(kp2[m.trainIdx].pt[1])), color)

cv2.imshow("result", img3)
cv2.waitKey()
```

Feature detection

```python
import numpy as np
import cv2

img1 = cv2.imread('church.jpg',0) # queryImage
img2 = cv2.imread('church_part.jpg',0) # trainImage

# Initiate SIFT detector
orb = cv2.ORB()

# find the keypoints and descriptors with SIFT
kp1, des1 = orb.detectAndCompute(img1,None)
kp2, des2 = orb.detectAndCompute(img2,None)

# create BFMatcher object
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)

# Match descriptors.
matches = bf.match(des1,des2)

# Sort them in the order of their distance.
matches = sorted(matches, key = lambda x:x.distance)

# Draw first 10 matches.
h1, w1 = img1.shape[:2]
h2, w2 = img2.shape[:2]
img3 = np.zeros((max(h1, h2), w1 + w2, 3), np.uint8)
img3[:h1, :w1, 0] = img1
img3[:h2, w1:, 0] = img2
img3[:, :, 1] = img3[:, :, 0]
img3[:, :, 2] = img3[:, :, 0]

for m in matches[:10]:
    # draw the keypoints
    # print m.queryIdx, m.trainIdx, m.distance
    color = tuple([np.random.randint(0, 255) for _ in xrange(3)])
    cv2.line(img3, (int(kp1[m.queryIdx].pt[0]), int(kp1[m.queryIdx].pt[1])) , (int(
kp2[m.trainIdx].pt[0] + w1), int(kp2[m.trainIdx].pt[1])), color)

cv2.imshow("result", img3)
cv2.waitKey()
```

Brute-force feature matching

```
import numpy as np
import cv2

img1 = cv2.imread('church.jpg',0) # queryImage
img2 = cv2.imread('church_part.jpg',0) # trainImage

# Initiate SIFT detector
orb = cv2.ORB()

# find the keypoints and descriptors with SIFT
kp1, des1 = orb.detectAndCompute(img1,None)
kp2, des2 = orb.detectAndCompute(img2,None)

# create BFMatcher object
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)

# Match descriptors.
matches = bf.match(des1,des2)

# Sort them in the order of their distance.
matches = sorted(matches, key = lambda x:x.distance)

# Draw first 10 matches.
h1, w1 = img1.shape[:2]
h2, w2 = img2.shape[:2]
img3 = np.zeros((max(h1, h2), w1 + w2, 3), np.uint8)
img3[:h1, :w1, 0] = img1
img3[:h2, w1:, 0] = img2
img3[:, :, 1] = img3[:, :, 0]
img3[:, :, 2] = img3[:, :, 0]

for m in matches[:10]:
    # draw the keypoints
    # print m.queryIdx, m.trainIdx, m.distance
    color = tuple([np.random.randint(0, 255) for _ in xrange(3)])
    cv2.line(img3, (int(kp1[m.queryIdx].pt[0]), int(kp1[m.queryIdx].pt[1])) , (int(
kp2[m.trainIdx].pt[0] + w1), int(kp2[m.trainIdx].pt[1])), color)

cv2.imshow("result", img3)
cv2.waitKey()
```

Display results

```
import numpy as np
import cv2

img1 = cv2.imread('church.jpg',0) # queryImage
img2 = cv2.imread('church_part.jpg',0) # trainImage

# Initiate SIFT detector
orb = cv2.ORB()

# find the keypoints and descriptors with SIFT
kp1, des1 = orb.detectAndCompute(img1,None)
kp2, des2 = orb.detectAndCompute(img2,None)

# create BFMatcher object
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)

# Match descriptors.
matches = bf.match(des1,des2)

# Sort them in the order of their distance.
matches = sorted(matches, key = lambda x:x.distance)

# Draw first 10 matches.
h1, w1 = img1.shape[:2]
h2, w2 = img2.shape[:2]
img3 = np.zeros((max(h1, h2), w1 + w2, 3), np.uint8)
img3[:h1, :w1, 0] = img1
img3[:h2, w1:, 0] = img2
img3[:, :, 1] = img3[:, :, 0]
img3[:, :, 2] = img3[:, :, 0]

for m in matches[:10]:
    # draw the keypoints
    # print m.queryIdx, m.trainIdx, m.distance
    color = tuple([np.random.randint(0, 255) for _ in xrange(3)])
    cv2.line(img3, (int(kp1[m.queryIdx].pt[0]), int(kp1[m.queryIdx].pt[1])) , (int(
kp2[m.trainIdx].pt[0] + w1), int(kp2[m.trainIdx].pt[1])), color)

cv2.imshow("result", img3)
cv2.waitKey()
```